Lab Manual

for

Linear Algebra

by

Jim Hefferon

Cover: my Chocolate Lab, Suzy.

Preface

This collection supplements the text *Linear Algebra*[1] with a number of explorations that help students solidify and extend their understanding of the subject, using the mathematical software Sage.[2]

A major goal of any undergraduate Mathematics program is to, over time, move students toward a higher-level, more abstract, grasp of the subject. For instance, most Calculus classes work on elaborate computations while later courses spend more effort on concepts and proofs, leaving less for the details of calculations. The text *Linear Algebra* fits into this development process.

Naturally the text presents its material using examples and practice problems that are small-sized and have manageable numbers: an assignment to multiply a pair of three by three matrices of small integers will build intuition, whereas asking students to do that same by-hand question with twenty by twenty matrices of ten decimal place numbers would be badgering. (And, even more worrisome, having students focus their intellectual energy on calculations instead of directing their attention to the ideas and proofs misleads them as to what the subject is about.)

However, mathematical software can mitigate this by extending the reach of what is reasonable to bigger systems, harder numbers, and computations that — while too much to do by hand — yield interesting information when they are done by a machine. For instance, an advantage of learning how to handle these tougher computations is that they are more like the ones that appear when students apply Linear Algebra to other subjects. Another advantage is that students see new ideas such as runtime growth measures.

Well then, why not teach straight from the computer system? Remember that we want to develop a higher-level understanding of the material, keeping the focus on vector spaces and linear maps. In our development the computations are a way to develop that understanding, not the main point. Some instructors may find that for their students this work is best left aside altogether, keeping a tight focus on the core material, while other instructors have students who will benefit.

Why Sage?

Sage is a very powerful mathematical software systems but so are many others. This manual uses it because it is Free[3] and Open Source[4] software.

In *Open Source Mathematical Software* [Joyner and Stein, 2007][5] the authors argue that for Mathematics using software that is Open Source is the best way forward.

[1] The text's home page http://joshua.smcvt.edu/linearalgebra has the PDF, the ancillary materials, and the LaTeX source. [2] See http://www.sagemath.org for the software and documentation. [3] The Free Software Foundation page http://www.gnu.org/philosophy/free-sw.html gives background and a definition. [4] See http://opensource.org/osd.html for a definition. [5] See http://www.ams.org/notices/200710/tx071001279p.pdf for the full text.

> Suppose Jane is a well-known mathematician who announces she has proved a theorem. We probably will believe her, but she knows that she will be required to produce a proof if requested. However, suppose now Jane says a theorem is true based partly on the results of software. The closest we can reasonably hope to get to a rigorous proof (without new ideas) is the open inspection and ability to use all the computer code on which the result depends. If the program is proprietary, this is not possible. We have every right to be distrustful, not only due to a vague distrust of computers but because even the best programmers regularly make mistakes.
>
> If one reads the proof of Jane's theorem in hopes of extending her ideas or applying them in a new context, it is limiting to not have access to the inner workings of the software on which Jane's result builds.

While professionals choose their tools by balancing many factors, their argument is persuasive. This manual uses Sage because it is very capable, including at Linear Algebra, because students can learn a great deal from it, and because it is Free.

This manual

This is Free. Get the latest version from http://joshua.smcvt.edu/linearalgebra. Also see that page for the license details and for the LaTeX source, including this manual.

I am glad to hear suggestions or corrections, especially from instructors who have class-tested the material. My contact information is on the same page.

The Sage output in this manual was generated automatically so it is sure to be accurate, except that I have (automatically) edited a few lines for length. My Sage identifies itself in this way.

```
1 'Sage Version 5.2, Release Date: 2012-07-25'
```

Acknowledgements

I am glad for this chance to thank the Sage Development Team for their work. In particular, without [Sage Development Team, 2012b] this manual would not have happened. I am glad also for the chance to mention [Beezer, 2011] as an inspiration.

Jim Hefferon
Mathematics, Saint Michael's College
Colchester, Vermont USA
2012-Sep-10

Contents

Vector Spaces

Sage can operate with vector spaces, for example by finding a basis for a space.

Real n-spaces

Start by creating a real vector space $\mathbb{R}^n$.

```
sage: V = RR^3
sage: V
Vector space of dimension 3 over Real Field with 53 bits of precision
```

You can test for membership.

```
sage: v1 = vector(RR, [1, 2, 3])
sage: v1 in V
True
sage: v2 = vector(RR, [1, 2, 3, 4])
sage: v2 in V
False
```

Consider the plane through the origin in $\mathbb{R}^3$ described by the equation $x - 2y + 2z = 0$. It is a subspace of $\mathbb{R}^3$ and we can describe it as a span.

$$W = \{ \begin{pmatrix} x \\ y \\ z \end{pmatrix} \Big| \, x = 2y - 2z \} = \{ \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} y + \begin{pmatrix} -2 \\ 0 \\ 1 \end{pmatrix} z \Big| \, y, z \in \mathbb{R} \}$$

You can create that subspace.

```
sage: V = RR^3
sage: v1 = vector(RR, [2, 1, 0])
sage: v2 = vector(RR, [-2, 0, 1])
sage: W = V.span([v1, v2])
```

And you can check that it is the desired space by doing some membership tests.

```
sage: v3 = vector(RR, [0, 1, 1])
sage: v3 in W
True
sage: v4 = vector(RR, [1, 0, 0])
sage: v4 in W
False
```

For a closer look at what's happening here, create a subspace of $\mathbb{R}^4$. (Some of the output in this chapter is edited to fit the page width.)

```
sage: V = RR^4
sage: V
Vector space of dimension 4 over Real Field with 53 bits of precision
sage: v1 = vector(RR, [2, 0, -1, 0])
sage: W = V.span([v1])
sage: W
Vector space of degree 4 and dimension 1 over Real Field with 53 bits
    of precision
Basis matrix:
[   1.00000000000000   0.000000000000000  -0.500000000000000
    0.000000000000000]
```

Sage has identified the dimension of the subspace and found a basis containing one vector (it prefers the vector with a leading 1). Sage also mentions the "Real Field" because we could take scalars from other number systems — in the book's final chapter the scalars are from the complex numbers — but we will here stick with real numbers. The "53 bits of precision" refers to the fact that Sage is using this computer's built-in model of real numbers.[1]

As earlier, the membership set relation works here.

```
sage: v2 = vector(RR, [2, 1, -1, 0])
sage: v2 in W
False
sage: v3 = vector(RR, [-4, 0, 2, 0])
sage: v3 in W
True
```

Basis Sage will give you a basis for your space.

```
sage: V=RR^2
sage: v=vector(RR, [1, -1])
sage: W = V.span([v])
sage: W.basis()
[
(1.00000000000000, -1.00000000000000)
]
```

Here is another example.

```
sage: V=RR^3
sage: v1 = vector(RR, [1, -1, 0])
sage: v2 = vector(RR, [1, 1, 0])
sage: W = V.span([v1, v2])
sage: W.basis()
[
(1.00000000000000, 0.000000000000000, 0.000000000000000),
(0.000000000000000, 1.00000000000000, 0.000000000000000)
]
sage: W.basis()
```

[1] This computer uses IEEE 754 double-precision binary floating-point numbers; if you have programmed then you may know this number model as binary64.

```
[
(1.00000000000000,  0.000000000000000,  0.000000000000000),
(0.000000000000000,  1.00000000000000,  0.000000000000000)
]
```

Adding a linearly dependent vector $\vec{v}_3$ to the spanning set doesn't change the space.

```
sage: v3 = vector(RR, [2, 3, 0])
sage: W_prime = V.span([v1, v2, v3])
sage: W_prime.basis()
[
(1.00000000000000,  0.000000000000000,  0.000000000000000),
(0.000000000000000,  1.00000000000000,  0.000000000000000)
]
```

In the prior example, notice that Sage does not simply give you back the linearly independent vectors that you gave it. Instead, Sage takes the vectors from the spanning set as the rows of a matrix, brings that matrix to reduced echelon form, and reports the nonzero rows (transposed to column vectors and so printed with parentheses) as the members of the basis. Because each matrix has one and only one reduced echelon form, each vector subspace of real n-space has one and only one such basis; this is a *canonical basis* for the space.

Sage will show you that it keeps the canonical basis as defining the space.

```
sage: W
Vector space of degree 3 and dimension 2 over Real Field with 53 bits
    of precision
Basis matrix:
[ 1.00000000000000  0.000000000000000  0.000000000000000]
[0.000000000000000   1.00000000000000  0.000000000000000]
sage: W_prime
Vector space of degree 3 and dimension 2 over Real Field with 53 bits
    of precision
Basis matrix:
[ 1.00000000000000  0.000000000000000  0.000000000000000]
[0.000000000000000   1.00000000000000  0.000000000000000]
```

If you are quite keen on your own basis then Sage will accomodate you.

```
sage: V = RR^3
sage: v1 = vector(RR, [1, 2, 3])
sage: v2 = vector(RR, [2, 1, 3])
sage: W = V.span_of_basis([v1, v2])
sage: W.basis()
[
(1.00000000000000,  2.00000000000000,  3.00000000000000),
(2.00000000000000,  1.00000000000000,  3.00000000000000)
]
sage: W
Vector space of degree 3 and dimension 2 over Real Field with 53 bits
    of precision
User basis matrix:
[1.00000000000000  2.00000000000000  3.00000000000000]
[2.00000000000000  1.00000000000000  3.00000000000000]
```

Space equality You can test spaces for equality.

```
sage: V = RR^4
sage: v1 = vector(RR, [1, 0, 0, 0])
sage: v2 = vector(RR, [1, 1, 0, 0])
sage: W12 = V.span([v1, v2])
sage: v3 = vector(RR, [2, 1, 0, 0])
sage: W13 = V.span([v1, v3])
```

Now use set membership to check that the two spaces W_{12} and W_{13} are equal.

```
sage: v3 in W12
True
sage: v2 in W13
True
```

Then, since obviously $\vec{v}_1 \in W_{12}$ and $\vec{v}_1 \in W_{13}$, the two spans are equal.

You can also just ask Sage.

```
sage: W12 == W13
True
```

Your check that equality works would be half-hearted if you didn't have an unequal case.

```
sage: v4 = vector(RR, [1, 1, 1, 1])
sage: W14 = V.span([v1, v4])
sage: v2 in W14
False
sage: v3 in W14
False
sage: v4 in W12
False
sage: v4 in W13
False
sage: W12 == W14
False
sage: W13 == W14
False
```

This illustrates a point about algorithms. Sage could check for equality of two spans by checking whether every member of the first spanning set is in the second space and vice versa (since the two spanning sets are finite). But Sage does something different. For each space it maintains the canonical basis and it checks for equality of the two spaces just by checking that they have the same canonical bases. These two algorithms have the same external behavior, in that both report correctly whether the two spaces are equal, but the second is faster. Finding the fastest way to do jobs is an important research area of computing.

Space operations Sage finds the intersection of two spaces. Consider these members of $\mathbb{R}^3$.

$$\vec{v}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad \vec{v}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad \vec{v}_3 = \begin{pmatrix} 0 \\ 0 \\ 3 \end{pmatrix}$$

We form two spans, the xy-plane $W_{12} = [\vec{v}_1, \vec{v}_2]$ and the yz-plane $W_{23} = [\vec{v}_2, \vec{v}_3]$. The intersection of these two is the y-axis.

```
sage: V=RR^3
sage: v1 =  vector(RR, [1, 0, 0])
sage: v2 =  vector(RR, [0, 1, 0])
sage: W12 = V.span([v1, v2])
sage: W12.basis()
[
(1.00000000000000, 0.000000000000000, 0.000000000000000),
(0.000000000000000, 1.00000000000000, 0.000000000000000)
]
sage: v3 = vector(RR, [0, 0, 2])
sage: W23 = V.span([v2, v3])
sage: W23.basis()
[
(0.000000000000000, 1.00000000000000, 0.000000000000000),
(0.000000000000000, 0.000000000000000, 1.00000000000000)
]
sage: W = W12.intersection(W23)
sage: W.basis()
[
(0.000000000000000, 1.00000000000000, 0.000000000000000)
]
```

Remember that the span of the empty set is the trivial space.

```
sage: v3 = vector(RR, [1, 1, 1])
sage: W3 = V.span([v3])
sage: W3.basis()
[
(1.00000000000000, 1.00000000000000, 1.00000000000000)
]
sage: W4 = W12.intersection(W3)
sage: W4.basis()
[

]
```

Sage will also find the sum of spaces, the span of their union.

```
sage: W5 = W12 + W3
sage: W5.basis()
[
(1.00000000000000, 0.000000000000000, 0.000000000000000),
(0.000000000000000, 1.00000000000000, 0.000000000000000),
(0.000000000000000, 0.000000000000000, 1.00000000000000)
]
sage: W5 == V
True
```

Other spaces

You can extend these computations to vector spaces that aren't a subspace of some $\mathbb{R}^n$. Just find a real space that matches the given one.

Consider this vector space set of quadratic polynomials (under the usual operations of polynomial addition and scalar multiplication).

$$\{a_2x^2+a_1x+a_0 \;\big|\; a_2-a_1=a_0\}=\{(a_1+a_0)x^2+a_1x+a_0 \;\big|\; a_1,a_0\in\mathbb{R}\}$$

It matches[1] this subspace of $\mathbb{R}^3$.

$$\{\begin{pmatrix}a_1+a_0\\a_1\\a_0\end{pmatrix} \;\big|\; a_1,a_0\in\mathbb{R}\}=\{\begin{pmatrix}1\\1\\0\end{pmatrix}a_1+\begin{pmatrix}1\\0\\1\end{pmatrix}a_0 \;\big|\; a_1,a_0\in\mathbb{R}\}$$

```
sage: V=RR^3
sage: v1 = vector(RR, [1, 1, 0])
sage: v2 = vector(RR, [1, 0, 1])
sage: W = V.span([v1, v2])
sage: W.basis()
[
(1.00000000000000, 0.000000000000000, 1.00000000000000),
(0.000000000000000, 1.00000000000000, -1.00000000000000)
]
```

Similarly you can represent this space of 2×2 matrices

$$\{\begin{pmatrix}a&b\\c&d\end{pmatrix} \;\big|\; a-b+c=0 \text{ and } b+d=0\}$$

by finding a real n-space just like it. Rewrite the given space

$$\{\begin{pmatrix}a&b\\c&d\end{pmatrix} \;\big|\; a=-c-d \text{ and } b=-d\}=\{\begin{pmatrix}-1&0\\1&0\end{pmatrix}c+\begin{pmatrix}-1&-1\\0&1\end{pmatrix}d \;\big|\; c,d\in\mathbb{R}\}$$

and then here is a natural matching real space.

```
sage: V = RR^4
sage: v1 = vector(RR, [-1, 0, 1, 0])
sage: v2 = vector(RR, [-1, -1, 0, 1])
sage: W = V.span([v1, v2])
sage: W.basis()
[
(1.00000000000000, -0.000000000000000, -1.00000000000000,
     -0.000000000000000),
(0.000000000000000, 1.00000000000000, 1.00000000000000,
     -1.00000000000000)
]
```

You could have gotten another matching space by going down the columns instead of across the rows, etc. But the important things about the space, such as its dimension, are unaffected by that choice.

[1] The textbook's chapter on Maps Between Spaces makes "matches" precise.

Maps

We've used Sage to define vector spaces. Next we explore operations that we can do on vector spaces.

Left/right

Sage represents linear maps differently than the book does. Rather than quarrel with our tools, below we will do it Sage's way.

Consider the application of a linear map to a member of a vector space $t(\vec{v})$. For example, with this function $t\colon \mathbb{R}^2 \to \mathbb{R}^3$ and this element of the domain

$$t(\begin{pmatrix} a \\ b \end{pmatrix}) = \begin{pmatrix} a+b \\ a-b \\ b \end{pmatrix} \qquad \vec{v} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$$

the map application gives this.

$$t(\begin{pmatrix} 1 \\ 3 \end{pmatrix}) = \begin{pmatrix} 4 \\ -2 \\ 3 \end{pmatrix}$$

To represent it we must fix bases. In this example we use the canonical bases $E_2 \subset \mathbb{R}^2$ and $E_3 \subset \mathbb{R}^3$. With respect to the bases the book finds a matrix $T = \mathrm{Rep}_{E_2,E_3}(t)$ and a vector $\vec{w} = \mathrm{Rep}_{E_2}(\vec{v})$, and represents $t(\vec{v})$ with the product $T\vec{w}$.

$$\begin{pmatrix} 1 & 1 \\ 1 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \end{pmatrix} = \begin{pmatrix} 4 \\ -2 \\ 3 \end{pmatrix}$$

That is, the book is write right: its notation puts the vector on the right of the matrix.

This choice is a matter of taste and many authors instead use a row vector that multiplies a matrix from the left. Sage falls in this camp. Sage represents the map application in this way.

$$\begin{pmatrix} 1 & 3 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 \\ 1 & -1 & 1 \end{pmatrix} = \begin{pmatrix} 4 & -2 & 3 \end{pmatrix}$$

Obviously the difference is cosmetic but can be confusing. The translation is that, compared to the book's representation $T\vec{w}$, Sage prefers the transpose $(T\vec{w})^{\mathsf{T}} = \vec{w}^{\mathsf{T}}\, T^{\mathsf{T}}$.

Defining

We will see two different ways to define a linear transformation.

Symbolically We first define a map that takes two inputs and returns three outputs.

```
sage: a, b = var('a, b')
sage: T_symbolic(a, b) = [a+b, a-b, b]
sage: T_symbolic
(a, b) |--> (a + b, a - b, b)
```

We have not yet defined a domain and codomain so this not a function — instead it is a prototype for a function. Make an instance of a function by applying $T_{symbolic}$ on a particular domain and codomain.

```
sage: T = linear_transformation(RR^2, RR^3, T_symbolic)
sage: T
Vector space morphism represented by the matrix:
[ 1.00000000000000  1.00000000000000  0.000000000000000]
[ 1.00000000000000 -1.00000000000000  1.00000000000000]
Domain: Vector space of dimension 2 over Real Field with 53 bits of
                                                                 precision
Codomain: Vector space of dimension 3 over Real Field with 53 bits of
                                                                 precision
```

Note the left/right issue again: Sage's matrix is the transpose of the matrix that the book would use.

Evaluating this function on a member of the domain gives a member of the codomain.

```
sage: v = vector(RR, [1, 3])
sage: T(v)
(4.00000000000000, -2.00000000000000, 3.00000000000000)
```

Sage can compute the interesting things about the transformation. Here it finds the null space and range space, using the equivalent terms *kernel* and *image*.

```
sage: T.kernel()
Vector space of degree 2 and dimension 0 over Real Field with 53 bits
                                                              of precision
Basis matrix:
[]
sage: T.image()
Vector space of degree 3 and dimension 2 over Real Field with 53 bits
                                                              of precision
Basis matrix:
[   1.00000000000000   0.000000000000000   0.500000000000000]
[ 0.000000000000000    1.00000000000000  -0.500000000000000]
```

The null space's basis is empty because it is the trivial subspace of the domain, with dimension 0. Therefore T is one-to-one.

The range space has a 2-vector basis. This agrees with the theorem that the dimension of the null space plus the dimension of the range space equals to the dimension of the domain.

For contrast consider a map that is not one-to-one.

```
sage: S_symbolic(a, b) = [a+2*b, a+2*b]
sage: S_symbolic
(a, b) |--> (a + 2*b, a + 2*b)
sage: S = linear_transformation(RR^2, RR^2, S_symbolic)
sage: S
Vector space morphism represented by the matrix:
[1.00000000000000 1.00000000000000]
[2.00000000000000 2.00000000000000]
Domain: Vector space of dimension 2 over Real Field with 53 bits of
                                                                  precision
Codomain: Vector space of dimension 2 over Real Field with 53 bits of
                                                                  precision
sage: v = vector(RR, [1, 3])
sage: S(v)
(7.00000000000000, 7.00000000000000)
```

This map is not one-to-one since the input $(a, b) = (2, 0)$ gives the same result as $(a, b) = (0, 1)$.

```
sage: S.kernel()
Vector space of degree 2 and dimension 1 over Real Field with 53 bits
                                                          of precision
Basis matrix:
[   1.00000000000000  -0.500000000000000]
sage: S.image()
Vector space of degree 2 and dimension 1 over Real Field with 53 bits
                                                          of precision
Basis matrix:
[1.00000000000000 1.00000000000000]
```

The null space has nonzero dimension, namely it has dimension 1, so Sage agrees that the map is not one-to-one.

Without looking at the range space we know that its dimension must be 1 because the dimensions of the null and range spaces add to the dimension of the domain. Again, Sage confirms our calculation.

Via matrices We can define a transformation by specifying the matrix representing its action.

```
sage: M = matrix(RR, [[1, 2], [3, 4], [5, 6]])
sage: m = linear_transformation(M)
sage: m
Vector space morphism represented by the matrix:
[1.00000000000000 2.00000000000000]
[3.00000000000000 4.00000000000000]
[5.00000000000000 6.00000000000000]
Domain: Vector space of dimension 3 over Real Field with 53 bits of
                                                                  precision
Codomain: Vector space of dimension 2 over Real Field with 53 bits of
                                                                  precision
```

Note again that Sage prefers the representation where the vector multiplies from the left.

```
sage: v = vector(RR, [7, 8, 9])
sage: m(v)
(76.0000000000000, 100.000000000000)
```

Sage has done this calculation.

$$\begin{pmatrix}7 & 8 & 9\end{pmatrix}\begin{pmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{pmatrix}=\begin{pmatrix}76 & 100\end{pmatrix}$$

If you have a matrix intended for a vector-on-the-right calculation (as in the book) then Sage will make the necessary adaptation.

```
sage: N = matrix(RR, [[1, 3, 5], [2, 4, 6]])
sage: n = linear_transformation(N, side='right')
sage: n
Vector space morphism represented by the matrix:
[1.00000000000000  2.00000000000000]
[3.00000000000000  4.00000000000000]
[5.00000000000000  6.00000000000000]
Domain: Vector space of dimension 3 over Real Field with 53 bits of
                                                                precision
Codomain: Vector space of dimension 2 over Real Field with 53 bits of
                                                                precision
sage: v = vector(RR, [7, 8, 9])
sage: n(v)
(76.0000000000000, 100.000000000000)
```

Although we gave it a `side='right'` option, the matrix that Sage shows by default is for `side='left'`.

Despite that we specified them differently, these two transformations are the same.

```
sage: m == n
True
```

We can ask the same questions of linear transformations created from matrices that we asked of linear transformations created from functions.

```
sage: m.kernel()
Vector space of degree 3 and dimension 1 over Real Field with 53 bits
                                                          of precision
Basis matrix:
[ 1.00000000000000  -2.00000000000000   1.00000000000000]
```

The null space of m is not the trivial subspace of $\mathbb{R}^3$ and so this function is not one-to-one. The domain has dimension 3 and the null space has dimension 1 and so the range space is a dimension 2 subspace of $\mathbb{R}^2$.

```
sage: m.image()
Vector space of degree 2 and dimension 2 over Real Field with 53 bits
                                                          of precision
Basis matrix:
[ 1.00000000000000  0.000000000000000]
[0.000000000000000   1.00000000000000]
sage: m.image() == RR^2
True
```

Sage lets us have the matrix represent a transformation involving spaces with nonstandard bases.

```
sage: M = matrix(RR, [[1, 2], [3, 4]])
sage: domain_basis = [vector(RR, [1, -1]), vector(RR, [1, 1])]
sage: D = (RR^2).subspace_with_basis(domain_basis)
sage: codomain_basis = [vector(RR, [2, 0]), vector(RR, [0, 3])]
sage: C = (RR^2).subspace_with_basis(codomain_basis)
sage: m = linear_transformation(D, C, M)
sage: m(vector(RR, [1, 0]))
(4.00000000000000, 9.00000000000000)
```

Sage has calculated that

$$\begin{pmatrix}1\\0\end{pmatrix} = (1/2)\begin{pmatrix}1\\-1\end{pmatrix} + (1/2)\begin{pmatrix}1\\1\end{pmatrix} \quad \text{so} \quad \text{Rep}_{domain\_basis}(\begin{pmatrix}1\\0\end{pmatrix}) = \begin{pmatrix}1/2\\1/2\end{pmatrix}$$

and then computed this.

$$\text{Rep}_{codomain\_basis}(m(\vec{v})) = \begin{pmatrix}1/2 & 1/2\end{pmatrix}\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix} = \begin{pmatrix}2 & 3\end{pmatrix} \quad \text{so} \quad m(\vec{v}) = 2\begin{pmatrix}2\\0\end{pmatrix} + 3\begin{pmatrix}0\\3\end{pmatrix} = \begin{pmatrix}4\\9\end{pmatrix}$$

Operations

Consider the set of all linear maps from some fixed vector space domain D to a fixed codomain C. The set of linear transformations has some natural operations, including addition and scalar multiplication. Sage can work with those operations.

Addition First create two functions.

```
sage: f_symbolic(x,y) = [x-y, x+2*y, 3*x]
sage: g_symbolic(x,y) = [y, 2*x-y, y]
```

We can add these two and with that new function make a linear transformation.

```
sage: f_symbolic+g_symbolic
(x, y) |--> (x, 3*x + y, 3*x + y)
sage: h = linear_transformation(RR^2, RR^3, f_symbolic+g_symbolic)
sage: h
Vector space morphism represented by the matrix:
[ 1.00000000000000  3.00000000000000  3.00000000000000]
[0.000000000000000  1.00000000000000  1.00000000000000]
Domain: Vector space of dimension 2 over Real Field with 53 bits of
                                                                precision
Codomain: Vector space of dimension 3 over Real Field with 53 bits of
                                                                precision
```

We could instead make two linear transformations and then add.

```
sage: f = linear_transformation(RR^2, RR^3, f_symbolic)
sage: g = linear_transformation(RR^2, RR^3, g_symbolic)
sage: h = f + g
sage: h
Vector space morphism represented by the matrix:
[ 1.00000000000000  3.00000000000000  3.00000000000000]
[0.000000000000000  1.00000000000000  1.00000000000000]
```

```
Domain: Vector space of dimension 2 over Real Field with 53 bits of
                                                                precision
Codomain: Vector space of dimension 3 over Real Field with 53 bits of
                                                                precision
```

Now we define the sum of the two by $F + G\,(\vec{v})$ is the map taking $\vec{v}$ to the sum of the two vectors $F(\vec{v})$ and $G(\vec{v})$.

```
sage: H = F+G
sage: F(v)+G(v)
(1.00000000000000, 3.00000000000000)
sage: H(v)
(1.00000000000000, 3.00000000000000)
```

The sum of two matrices is defined as the operation on the representations that yields the representation of the function sum.

```
sage: F.matrix(side='right')
[1.00000000000000 2.00000000000000]
[3.00000000000000 4.00000000000000]
sage: G.matrix(side='right')
[ 1.00000000000000 -1.00000000000000]
[ 2.00000000000000 -2.00000000000000]
sage: H.matrix(side='right')
[2.00000000000000 1.00000000000000]
[5.00000000000000 2.00000000000000]
```

The scalar multiple of a function and the scalar multiple of a matrix are defined analgously, except that we must write the scalar on the right.

```
sage: F(v)
(-1.00000000000000, -1.00000000000000)
sage: H = F*3
sage: H(v)
(-3.00000000000000, -3.00000000000000)
```

Composition

Inverse

Alternate bases

Bibliography

Robert A. Beezer. Sage for Linear Algebra. http://linear.ups.edu/download/fcla-2.22-sage-4.7.1-preview.pdf, 2011.

Jim Hefferon. Linear Algebra. http://joshua.smcvt.edu/linearalgebra, 2012.

David Joyner and William Stein. Open source mathematical software. *Notices of the AMS*, page 1279, November 2007.

Sage Development Team. Sage tutorial 5.3. http://www.sagemath.org/pdf/SageTutorial.pdf, 2012a.

Sage Development Team. Sage reference manual 5.3. http://www.sagemath.org/pdf/reference.pdf, 2012b.